Submitted by:	
Marker:	

Notes	Smark	ratio	letter		
	100%	100.00%		**1 Part 1 - Map**	1 -
	100%	50.00%		**1.1 Knowledge/Correctness of Game Rules**	1.1 -
	100%	5.00%	A	1.1.1 Students are fully aware of the correct Warzone game rules to implement during the presentation	1.1.1 -
	100%	5.00%	A	1.1.2 Code is implementing game mechanics that is fully according the Warzone game	1.1.2 -
	100%	5.00%	A	**1.1.3 Compliance of solution with Stated Problem**	1.1.3 -
	100%	5.00%	A	1.1.4 Territories are associated with players	1.1.4 -
	100%	5.00%	A	1.1.5 Territories contain armies	1.1.5 -
	100%	5.00%	A	1.1.6 Presence of a Map::validate() method	1.1.6 -
	100%	5.00%	A	1.1.7 Map is a collection of Territories, Continents are a subset of the Territories in the graph	1.1.7 -
	100%	5.00%	A	1.1.8 Map can be made to represent any map configuration using a graph data structure with Territories as nodes, and pointers to Territories as edges	1.1.8 -
	100%	5.00%	A	1.1.9 All data members of user-defined class type are of pointer type	1.1.9 -
	100%	5.00%	A	1.1.10 Driver clearly demonstrates that Map::validate() method works properly. Absence of Joiner functions	1.1.10 -
	100%	5.00%	A	1.1.11 All classes implement a correct copy constructor, assignment operator, and dtoren interface operator	1.1.11 -
	100%	5.00%	A	1.1.12 Absence of memory leaks	1.1.12 -
	100%	5.00%	A	1.1.13 Driver clearly demonstrates that the Map::validate() method works that 1) a map is a connected graph, 2) a continent is a connected subgraph, 3) each country belongs to one and only one continent	1.1.13 -
	100%	5.00%	A	1.1.14 Driver clearly demonstrates that various valid/invalid Domination maps files can be read, resulting in the creation and validation of different maps	1.1.14 -
	100%	50.00%		**1.2 Modularity of Solution**	1.2 -
	100%	5.00%	A	1.2.1 All of is implemented in the file data named Map.cpp/Map.h, and no other files.	1.2.1 -
	100%	5.00%	A	1.2.2 Presence of classes named Territory, Map, and Maphandler, usage of class Player to signify Territory ownership	1.2.2 -
	100%	50.00%		**1.3 Mastery of Language/Tools/Libraries**	1.3 -
	100%	5.00%	A	1.3.1 Pre-progress value created during the demonstration or code review	1.3.1 -
	100%	5.00%	A	1.3.2 Students were very clear in technical discussions during the demonstration	1.3.2 -
	100%	50.00%		**1.3.3 Code readability: name conventions, clarity of code, use of comments**	1.3. -
	100%	5.00%	A	1.3.3.1 All user-defined classes, methods, free functions, and operators are documented	1.3.3.1 -
	100%	5.00%	A	1.3.3.2 Absence of commented-out code	1.3.3.2 -

	100%	100.00%		**2 Part 2 - Player**	2 -
	100%	50.00%		**2.1 Knowledge/Correctness of Game Rules**	2.1 -
	100%	5.00%	A	2.1.1 Students are fully aware of the correct Warzone game rules to implement during the presentation	2.1.1 -
	100%	5.00%	A	2.1.2 Code is implementing game mechanics that is fully according the Warzone game	2.1.2 -
	100%	50.00%		**2.2 Compliance of solution with Stated Problem**	2.2 -
	100%	8.67%	A	2.2.1 A player includes a collection of territories - using the Territory class	2.2.1 -
	100%	8.67%	A	2.2.2 A player includes a hand of cards - using the Hand class	2.2.2 -
	100%	8.67%	A	2.2.3 A player includes a list of orders - using the OrderList class	2.2.3 -
	100%	8.67%	A	2.2.4 Player class includes issueOrder() and toDefend() and toAttack() methods that create a list of orders	2.2.4 -
	100%	8.67%	A	2.2.5 Driver clearly demonstrates that issueOrder() and toDefend() and toAttack() methods put orders (i.e) the list of orders of the player.	2.2.5 -
	100%	8.67%	A	2.2.6 All data members of user-defined class type are of pointer type	2.2.6 -
	100%	8.67%	A	2.2.7 Classes declared in header file. Functions implementation signifies. Absence of Joiner functions	2.2.7 -
	100%	8.67%	A	2.2.8 All classes implement a correct copy constructor, assignment operator, and dtoren interface operator	2.2.8 -
	100%	8.67%	A	2.2.9 Absence of memory leaks	2.2.9 -
	100%	8.67%	A	2.2.10 Driver clearly demonstrates that IssueTask and IssueNext can be called and return a list of Territories	2.2.10 -
	100%	8.67%	A	2.2.11 Driver clearly demonstrates that IssueOrder can be called and will check if an order is to be placed in the list of orders of the player	2.2.11 -
	100%	8.67%	A	2.2.12 Driver clearly demonstrates that a player owns a hand of cards	2.2.12 -
	100%	8.67%	A	2.2.13 Driver clearly demonstrates that a player owns a list of owned territories	2.2.13 -
	100%	50.00%		**2.3 Modularity of Solution**	2.3 -
	100%	5.00%	A	2.3.1 All of is implemented in the file data named Player.cpp/Player.h	2.3.1 -
	100%	5.00%	A	2.3.2 Presence of Player class, usage of classes Hand and OrderList	2.3.2 -
	100%	50.00%		**2.4 Mastery of Language/Tools/Libraries**	2.4 -
	100%	5.00%	A	2.4.1 Pre-progress value created during the demonstration or code review	2.4.1 -
	100%	5.00%	A	2.4.2 Students were very clear in technical discussions during the demonstration	2.4.2 -
	100%	50.00%		**2.4.3 Code readability: name conventions, clarity of code, use of comments**	2.4. -
	100%	5.00%	A	2.4.3.1 All user-defined classes, methods, free functions, and operators are documented	2.4.3.1 -
	100%	5.00%	A	2.4.3.2 Absence of commented-out code	2.4.3.2 -

	100%	100.00%		**3 Part 3 - Orders**	3 -
	100%	50.00%		**3.1 Knowledge/Correctness of Game Rules**	3.1 -
	100%	5.00%	A	3.1.1 Students are fully aware of the correct Warzone game rules to implement during the presentation	3.1.1 -
	100%	5.00%	A	3.1.2 Code is implementing game mechanics that is fully according the Warzone game	3.1.2 -
	100%	50.00%		**3.2 Compliance of solution with Stated Problem**	3.2 -
	100%	5.00%	A	3.2.1 Orders of any subclass can be created by the player and placed sequentially in the list of orders	3.2.1 -
	100%	5.00%	A	3.2.2 Orders can be moved in the list using the move() method	3.2.2 -
	100%	5.00%	A	3.2.3 Orders can be removed from the list using the remove() method	3.2.3 -
	100%	5.00%	A	3.2.4 Orders have an execute() method	3.2.4 -
	100%	5.00%	A	3.2.5 All data members of user-defined class type are of pointer type	3.2.5 -
	100%	5.00%	A	3.2.6 Classes declared in header file. Functions implementation signifies. Absence of Joiner functions	3.2.6 -
	100%	5.00%	A	3.2.7 All classes implement a correct copy constructor, assignment operator, and dtoren interface operator	3.2.7 -
	100%	5.00%	A	3.2.8 Absence of memory leaks	3.2.8 -
	100%	5.00%	A	3.2.9 Driver clearly demonstrates that orders of every kind can be created and placed in an OrderList object	3.2.9 -
	100%	5.00%	A	3.2.10 Driver clearly demonstrates that orders can be moved and deleted using move() and remove()	3.2.10 -
	100%	5.00%	A	3.2.11 Driver clearly demonstrates that orders can be validated and executed using validate() and execute()	3.2.11 -
	100%	50.00%		**3.3 Modularity of Solution**	3.3 -
	100%	5.00%	A	3.3.1 All of is implemented in the file data Orders.h/Orders.cpp	3.3.1 -
	100%	5.00%	A	3.3.2 Presence of OrderList, and Order classes. Presence of Deploy, Advance, Bomb, Blockade, Airlift, and Negotiate as subclasses of the Order class	3.3.2 -
	100%	50.00%		**3.4 Mastery of Language/Tools/Libraries**	3.4 -
	100%	5.00%	A	3.4.1 Pre-progress value created during the demonstration or code review	3.4.1 -
	100%	5.00%	A	3.4.2 Students were very clear in technical discussions during the demonstration	3.4.2 -
	100%	50.00%		**3.4.3 Code readability: name conventions, clarity of code, use of comments**	3.4. -
	100%	5.00%	A	3.4.3.1 All user-defined classes, methods, free functions, and operators are documented	3.4.3.1 -
	100%	5.00%	A	3.4.3.2 Absence of commented-out code	3.4.3.2 -

	100%	100.00%		**4 Part 4 - Cards decks/hand**	4 -
	100%	50.00%		**4.1 Knowledge/Correctness of Game Rules**	4.1 -
	100%	5.00%	A	4.1.1 Students are fully aware of the correct Warzone game rules to implement during the presentation	4.1.1 -
	100%	5.00%	A	4.1.2 Code is implementing game mechanics that is fully according the Warzone game	4.1.2 -
	100%	80.00%		**4.2 Compliance of solution with Stated Problem**	4.2 -
	100%	4.00%	A	4.2.1 Deck has a method draw() that removes the card from the Deck and returns that card	4.2.1 -
	100%	4.00%	A	4.2.2 Deck has a method add() that adds a card from the card from the Deck and returns that card	4.2.2 -
	100%	4.00%	A	4.2.3 Hand contains a collection of Cards	4.2.3 -
	100%	4.00%	A	4.2.4 Card has a play() method, which creates an Order object and places it in the Players' list of orders, removes the card from the Hand, and places it back in the Deck.	4.2.4 -
	100%	4.00%	A	4.2.5 All data members of user-defined class type are of pointer type	4.2.5 -
	100%	4.00%	A	4.2.6 Classes declared in header file. Functions implementation signifies. Absence of Joiner functions	4.2.6 -
	100%	4.00%	A	4.2.7 All classes implement a correct copy constructor, assignment operator, and dtoren interface operator	4.2.7 -
	100%	4.00%	A	4.2.8 Absence of memory leaks	4.2.8 -
	100%	4.00%	A	4.2.9 Driver clearly demonstrates that draw() effectively removes the known card from the deck and returns it (i.e)	4.2.9 -
	100%	4.00%	A	4.2.10 Driver clearly demonstrates that play() creates an Order object and places it in the Players' list of orders, removes the card from the Hand, and places it back in the Deck.	4.2.10 -
	100%	50.00%		**4.3 Modularity of Solution**	4.3 -
	100%	5.00%	A	4.3.1 All is implemented in the file data Cards.h/Cards.cpp	4.3.1 -
	100%	5.00%	A	4.3.2 Presence of Card, Deck, and Hand classes; usage of classes Player, Order, and OrderList	4.3.2 -
	100%	50.00%		**4.4 Mastery of Language/Tools/Libraries**	4.4 -
	100%	5.00%	A	4.4.1 Pre-progress value created during the demonstration or code review	4.4.1 -
	100%	5.00%	A	4.4.2 Students were very clear in technical discussions during the demonstration	4.4.2 -
	100%	50.00%		**4.4.3 Code readability: name conventions, clarity of code, use of comments**	4.4. -
	100%	5.00%	A	4.4.3.1 All user-defined classes, methods, free functions, or operators are documented	4.4.3.1 -
	100%	5.00%	A	4.4.3.2 Code is clear and there is zero presence of commented-out code	4.4.3.2 -

	100%	100.00%		**5 Part 5 - Game engine**	5 -
	100%	50.00%		**5.1 Knowledge/Correctness of Game Rules**	5.1 -
	100%	5.00%	A	5.1.1 Students are fully aware of the correct Warzone game rules to implement during the presentation	5.1.1 -
	100%	5.00%	A	5.1.2 Code is implementing game mechanics that is fully according the Warzone game	5.1.2 -
	100%	50.00%		**5.2 Compliance of solution with Stated Problem**	5.2 -
	100%	4.00%	A	5.2.1 The state of the game is held in a data member of the GameEngine class	5.2.1 -
	100%	4.00%	A	5.2.2 Console input is used to get commands from the user	5.2.2 -
	100%	4.00%	A	5.2.3 Depending on the command and the state, a transition is made to another state by changing the value held by the state variable	5.2.3 -
	100%	4.00%	A	5.2.4 If the user enter an invalid command for the current state, the command is rejected and an error message is output to the console.	5.2.4 -
	100%	4.00%	A	5.2.5 The state transitions are implemented exactly as depicted in the assignment handout.	5.2.5 -
	100%	4.00%	A	5.2.6 Classes declared in header file. Functions implementation signifies.	5.2.6 -
	100%	4.00%	A	5.2.7 Classes declared in header file. Functions implementation signifies. Absence of Joiner functions	5.2.7 -
	100%	4.00%	A	5.2.8 All classes implement a correct copy constructor, assignment operator, and dtoren interface operator	5.2.8 -
	100%	4.00%	A	5.2.9 Absence of memory leaks	5.2.9 -
	100%	4.00%	A	5.2.10 Driver clearly demonstrates that commands can be used to navigate through the different states as depicted in the assignment handout.	5.2.10 -
	100%	50.00%		**5.3 Modularity of Solution**	5.3 -
	100%	5.00%	A	5.3.1 Implemented in a file due named GameEngine.cpp/GameEngine.h	5.3.1 -
	100%	5.00%	A	5.3.2 Presence of classes State, Transition	5.3.2 -
	100%	50.00%		**5.4 Mastery of Language/Tools/Libraries**	5.4 -
	100%	5.00%	A	5.4.1 Pre-progress value created during the demonstration or code review	5.4.1 -
	100%	5.00%	A	5.4.2 Students were very clear in technical discussions during the demonstration	5.4.2 -
	100%	50.00%		**5.4.3 Code readability: name conventions, clarity of code, use of comments**	5.4. -
	100%	5.00%	A	5.4.3.1 All user-defined classes, methods, free functions, or operators are documented	5.4.3.1 -
	100%	5.00%	A	5.4.3.2 Code is clear and there is zero presence of commented-out code	5.4.3.2 -

Marker Instructions:

Enter values only in the red cells. Everything else is calculated automatically

In Column D, enter either A, B, C, or F for each marking element

Enter notes in column A

If there is an entire part that has been agreed to not be graded, delete one of the columns in the green table (Part 1 to Part 5)

Designer Instructions:

Add/remove some time with red cells to add/remove evaluation criteria

When you add lines, add them only in the middle of the area, or else the calculation is going to be wrong

To add lines, add a new row, then copy-etc-t-ove of the existing rows.

To remove parts, remove the entire block above, then remove the corresponding column in the green table to the right

To add more than 5 parts, it's more complicated.

Grading rubric	Letter	Part 1	Part 2	Part 3	Part 4	Part 5	Score ratio	Total marks
Knowledge/Correctness of Game Rules	A	2.00	2.00	2.00	2.00	2.00	1.00	2.00
Compliance of solution with Stated Problem	A	2.00	2.00	2.00	2.00	2.00	1.00	2.00
Modularity of Solution	A	2.00	2.00	2.00	2.00	2.00	1.00	2.00
Mastery of Language/Tools/Libraries	A	2.00	2.00	2.00	2.00	2.00	1.00	2.00
Code readability: name conventions, clarity of code, use of comments	A	2.00	2.00	2.00	2.00	2.00	1.00	2.00
Total	30	6.00	6.00	6.00	6.00	6.00		30.00

Final marks